

Steven M. Hoffberg

From: Steven M. Hoffberg [steve@hoffberg.org]
Sent: Thursday, November 18, 2004 12:21 PM
To: 'Nguyen, Nga'
Subject: 09/599,163 http_clickshare.c

```

/* -----
 * http_clickshare.c: routines for doing Clickshare authentication and
 * session tracking
 *
 * M Callahan <michael@Newshare.com>
 * D Oliver <dave@Newshare.com>
 * Copyright 1995 Newshare Inc.
 * -----
 */

#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/un.h>
#include <sys/wait.h>
#include <string.h>
#ifdef SOLARIS2
#define index(a,b) strchr(a,b)
#include <sys/time.h>
#include <sys/resource.h> /* to get getrlimit() */
#endif /* SOLARIS2 */
#include <gdbm.h>
#include <stdio.h>

#include "httpd.h"

#include "tv_s_client.h" /* interface to Clickshare auth facility */
#include "tv_s_config.h" /* interface to clickshare.conf reader */
#include "tv_s_error.h" /* error codes for tv_s_validate_token */
#include "../reg/user_db.h"

/* -----
 * definitions
 * -----
 */

/* how to contact Clickshare Corporation */

```

```

#define CLICKSHARE_SERVER "www.clickshare.com" /* hostport part only */

/* local pages that assist users */

#define BAD_TOKEN_PAGE      "/click_user/bad_token.html"
#define NEW_USER_HELP_PAGE  "/click_user/welcome.html"
#define LOGOUT_THANKYOU_PAGE "/click_user/logout_thankyou.html"
#define LOGOUT_ERROR_PAGE   "/click_user/logout_error.html"
#define BAD_TESTDRIVE_PAGE  "/click_user/bad_testdrive.html"

/* a test-drive user "jumps" into content from this page (which is probably
 * a link to a real page).
 */

#define TESTDRIVE_JUMP_PAGE  "/click_user/testdrive_jump.html"

/* well known location at each PM to contact to re-auth */

#define TOKEN_TIMEOUT_PAGE   "/auth/auth.html"

/* CGI script which generates a page when a user attempts access
 * to a CS-protected page without a token
 */
#define AUTH_REQUIRED_PAGE   "/click_cgi/auth_required"

#ifdef LINUX
/* for NR_OPEN */
#include <linux/fs.h>
#endif

#include "http_clickshare.h"

/* -----
 * variables used here or exported
 * -----
 */

/*
 * socket used for logging
 */

static int cs_log_socket = -1;

/* clickshare_dont_shutdown: nonzero means that we have a child
 * process doing token addition, so don't call shutdown(2) on the socket
 */

int clickshare_dont_shutdown = 0;

/* clickshare_saved_url, clickshare_saved_query: we shove the url
 * and (optional) query string from the url when we get a chance

```

```

/* in clickshare_process_args for logging purposes later
*/

static char *clickshare_saved_url;
static char *clickshare_saved_query;

/* clickshare_redirect_url: if we get a REDIRECT= query string, we
 * stuff it here
*/

static char *clickshare_redirect_url;

/* parameters given in clickshare.conf file:
 * AddTokenScript, LoggerFile, RegistrationDB, UseridDB
*/

static char *clickshare_add_token_script = (char *) NULL;
static char *clickshare_logger_file      = (char *) NULL;
static char *clickshare_registrationDB   = (char *) NULL;
static char *clickshare_useridDB        = (char *) NULL;

extern char msgString[];

static TVS_SERVER tvs_server = (TVS_SERVER) NULL;
static TVS_PROFILE tvs_profile = (TVS_PROFILE) NULL;
unsigned int clickshare_pageclass;

/* tvs_login_token: nonzero means we had magic token "TVS=login"
 * requesting login
*/
static int tvs_login_token = 0;

void clickshare_after_auth_redirect (char *, FILE *);
void clickshare_log_hello();

#ifdef SOLARIS2
/* -----
 * replacement for getdtablesize()
 * -----
*/

int
getdtablesize()
{
    struct rlimit rlp;

    if (getrlimit(RLIMIT_NOFILE, &rlp) < 0) return 1;
    else
        return (int) rlp.rlim_max;
}
#endif /* SOLARIS2 */

```

```

/* -----
 * check for errors on auth
 * -----
 */

int clickshare_token_ok() {
    return tvs_profile != NULL;
}

/* -----
 * indicate whether we got the magic token
 * -----
 */

int clickshare_login_token() {
    return tvs_login_token;
}

/* -----
 * open up the clickshare service and get site-definable params
 * -----
 */

void clickshare_open(char *conf) {
    if (tvs_server)
        return;

    tvs_server = tvs_initialize_service(conf);
    if (!tvs_server) {
        sprintf(msgString, "httpd: unable to initialize Clickshare\n");
        LogMsg (LOG_ERR, msgString);
        exit (1);
    }

    /*
     * get parameters, sanity checking as we go along
     */

    clickshare_add_token_script = (char *) tvs_get_config_param ("AddTokenScript");

    if (!clickshare_add_token_script) {
        LogMsg (LOG_ERR, "no AddTokenScript parameter");
        exit (1);
    }

    if (access (clickshare_add_token_script, X_OK) != 0) {
        sprintf (msgString, "cannot execute AddTokenScript script %s",
            clickshare_add_token_script);
        LogMsg (LOG_ERR, msgString);
        exit(1);
    }
}

```

```

}

clickshare_logger_file = tvs_get_config_param ("LoggerFile");

if (!clickshare_logger_file)
    LogMsg (LOG_WARNING, "no LoggerFile parameter specified: CS logging turned
off");

clickshare_registrationDB = tvs_get_config_param ("RegistrationDB");

if (!clickshare_registrationDB) {
    LogMsg (LOG_ERR, "no RegistrationDB parameter specified");
    exit(1);
}

if (access (clickshare_registrationDB, R_OK) != 0) {
    sprintf (msgString, "cannot read RegistrationDB %s",
            clickshare_registrationDB);
    LogMsg (LOG_ERR, msgString);
    exit(1);
}

/* presently, httpd does not need to access UseridDB itself */
clickshare_useridDB = tvs_get_config_param ("UseridDB");

if (!clickshare_useridDB) {
    LogMsg (LOG_WARNING, "warning: no UseridDB parameter specified");
}

if (access (clickshare_useridDB, R_OK) != 0) {
    sprintf (msgString, "warning: cannot read UseridDB %s",
            clickshare_useridDB);
    LogMsg (LOG_WARNING, msgString);
}

/* register with the logging facility */

clickshare_log_hello();
}

/* -----
 * drop out of clickshare session
 * -----
 */

void clickshare_terminate() {
    tvs_drop_service();
    tvs_server = NULL;
}

```

```

/*-----
 * authenticate a new user from the local auth database
 *-----
*/

void clickshare_authenticate_user(char *user, char *password, FILE *out) {
    GDBM_FILE userdb;
    REG_PROFILE profile;
    static char encrypted_password[100]; /* XXX */
    int tries;
    char errstr[1000];
    char *tok, *p;

    if (!clickshare_registrationDB)
        return;

    for (tries = 0; tries < 60; tries++) {
        /* open_name_db silently picks up the RegistrationDB parameter */
        userdb = open_name_db(GDBM_READER);
        if (userdb)
            break;
        sleep(1);
    }

    if (!userdb)
        die (SERVER_ERROR, "cannot open registration database", out);

    profile = name_get(userdb, user);

    close_name_db(userdb);

    if (!profile) {
        sprintf(errstr, "Cannot find user %s in registration database",
            user);
        auth_bong(errstr, out);
    }

    /* password as stored in the profile is unencrypted.
       in order to be compatible with what caller expects,
       crypt it now. */
    if (strcmp(profile->clickshare_password, password))
        auth_bong("Incorrect password", out);

    /* OK, we have a user!
       * Construct a user profile.
       */

    tvs_profile = tvs_make_user_profile();
    tvs_login_token = 0;

```

```

/* set the non-default values */

tvsv_set_pmid(tvsv_profile, tvsv_pm_id);
tvsv_set_userid(tvsv_profile, profile->clickshare_userid);
tvsv_set_hostid(tvsv_profile, inet_addr(remote_ip));
tvsv_set_sessionid(tvsv_profile, tvsv_make_sessionid(tvsv_pm_id, 0));

/* use values from user preference database */

tvsv_set_adv_context(tvsv_profile, profile->pref_advertising_level);
tvsv_set_privacy1_flag(tvsv_profile, profile->pref_privacy1);
tvsv_set_pdac_flag(tvsv_profile, profile->pref_parental_discretion);
tvsv_set_premium_flag(tvsv_profile, profile->pref_premium_charges);

free(profile);

tok = tvsv_new_token(tvsv_profile);
if (!tok)
    die(SERVER_ERROR, "cannot construct TVS token", out);

p = (char *) malloc(strlen(tok) + 5);
if (!p)
    die(NO_MEMORY, "constructing TVS token", out);

strcpy(p, "TVS=");
strcpy(&p[4], tok);

if (clickshare_redirect_url) {
    /* this was a re-auth... bounce the user back to the page he wants */
    char buf[2000];

    strcpy(buf, clickshare_redirect_url + strlen("REDIRECT="));
    unescape_url(buf);

    /* does it already have a query component? */
    if (index(buf, '?'))
        strcat(buf, "+");
    else
        strcat(buf, "?");

    /* add TVS=token */
    strcat(buf, p);
    die(REDIRECT, buf, out);
}

clickshare_after_auth_redirect(p, out);
}

/* -----
* based on the saved URL and query options, send a redirect with the TVS token
* attached

```

```

* -----
*/

void clickshare_after_auth_redirect(char *tok, FILE *out) {
    char buf[2000];

    /* get http://ourhost:ourport/url */
    construct_url (buf, clickshare_saved_url);
    /* escape this -- the things we add later are already escaped */
    escape_url (buf);
    /* add token */
    strcat (buf, "?");
    strcat (buf, tok);
    /* add old query */
    if (clickshare_saved_query && *clickshare_saved_query) {
        strcat (buf, "+");
        strcat (buf, clickshare_saved_query);
    }

    die(REDIRECT, buf, out);
}

/* -----
* get the TVS auth token out of a URL
* -----
*/

char *clickshare_extract_token (char *args, char *param) {
    char *p, *q, *tvb_buf;
    int len;

    if (!args)
        return NULL;

    /* find param as a parameter in the query string args */
    len = strlen(param);
    p = args;
    while (*p) {
        if (!strcmp(p, param, len))
            break;
        p = strchr(p, '+');
        if (!p)
            return NULL;
        p++;
    }

    if (!*p)
        return NULL;

    /* got TVS: copy it out of string */
    /* is it terminated by a "+"? */

```



```

q = index(p, '+');
if (q) {
    /* yes: copy up to "+" character into tvs_buf */
    len = q - p;
    tvs_buf = (char *) malloc(len + 1);
    strcpy(tvs_buf, p, len);
    tvs_buf[len] = '\0';

    /* now close up arg string by copying everything
       past the "+" character over the "TVS=" stuff */
    strcpy(p, q + 1);
} else {
    /* no: TVS argument ends with a null, so dup it */
    tvs_buf = strdup(p);

    /* close up arg string by zapping a null over the "T" of "TVS" */
    *p = '\0';

    /* if the string isn't now empty, make sure we remove a trailing
       "+", if any */
    if (p != args && p[-1] == '+')
        p[-1] = '\0';
}

return tvs_buf;
}

/* -----
 * wait when we exit (so that kids dont get death signals)
 * -----
 */

void clickshare_wait_at_exit () {
    int status;

    /* wait for child before exiting */
    wait(&status);
    sleep (50);
}

/* -----
 * wrapper to handle errors in authentication gracefully
 * -----
 */

static TVS_PROFILE
clickshare_attempt_validation(char *token, FILE *out)
{
    TVS_PROFILE prof;
    char _where[MAX_STRING_LEN], *p;

```

```

/* Here we determine if the customer at the other end of the pipe is
 * currently valid. If user is not valid, we ship him back to a
 * variety of locations, depending on how much we know about him.
 */

/* NOTE: remote_ip coming to us from http_request.c (global) */

prof = tvs_validate_token(token, inet_addr(remote_ip));

if (!prof) {
    switch((int) tvs_get_token_error_type()) {

        /* handle these cases locally */
        case TVS_NO_TOKEN : /* no token attached */
        case TVS_TOKEN_IS_INVALID : /* token invalid */
/**    sprintf(_where, "http://%s/%s", CLICKSHARE_SERVER, NEW_USER_HELP_PAGE);**/
        construct_url(_where, NEW_USER_HELP_PAGE);
        die(REDIRECT, _where, out);

        /* redirect to home publisher for re-authorization */
        case TVS_TOKEN_TIMED_OUT : /* token timeout */
        sprintf(_where, "http://%s/%s?REDIRECT=",
            tvs_get_user_home(), TOKEN_TIMEOUT_PAGE);
        /* now add old URL, _escaped_ */
        p = &_where[strlen(_where)];
        construct_url(p, clickshare_saved_url);
        p = &_where[strlen(_where)];
        if (clickshare_saved_query && *clickshare_saved_query) {
            *p++ = '?';
            strcpy(p, clickshare_saved_query);
        }
        escape_url(p);
        die(REDIRECT, _where, out);

        case TVS_TOKEN_IS_OK : /* valid user? this is messy ! */
        LogMsg(LOG_ERR, "error from tvs_request_validation not setting TVS_ERROR");
        /* and fall thru ... */
        /* give these errors to the clickshare boys */
        case TVS_TOKEN_IS_GARBAGE : /* really messy token */
        case TVS_USER_AT_INVALID_HOST : /* invalid host/user */
        default :
            sprintf(_where, "http://%s/%s", CLICKSHARE_SERVER, BAD_TOKEN_PAGE);
            die(REDIRECT, _where, out);
        }
    }

    return prof;
}

```

```

/*-----
 * make some sense of a user's request, auth if required, bounce if bad
 *-----
*/

void clickshare__process_args (char *url, char *args, FILE *outf) {
    char *tok;

    /* assume we don't have a token */
    clickshare_dont_shutdown = 0;
    tvs_profile = NULL;
    tvs_login_token = 0;
    clickshare_pageclass = -1;
    clickshare_redirect_url = NULL;

    tok = clickshare_extract_token (args, "TVS=");

    clickshare_saved_url = url ? strdup(url) : "";
    clickshare_saved_query = args ? strdup(args) : "";

    if (strcmp(url, TOKEN_TIMEOUT_PAGE) == 0) {
        /* we have a redirect here */
        clickshare_redirect_url = clickshare_extract_token (args, "REDIRECT=");

        if (clickshare_redirect_url) {
            /* we definitely want to prompt for username/password,
             not put up the explanatory page */
            tvs_login_token = 1;
            if (tok)
                free (tok);
            tok = NULL;
            return;
        }
        /* oops! shouldn't get here: that would mean we're
         going to the token revalidation page without a redirect
         target--Dave will have to pay for somebody's beer!
        */
    }

    if (!tok) {
        return;          /* nothing to do */
    }

    /* check for special "TVS=login" token to indicate a desire
     * to log in
     */
    if (strcmp(tok, "TVS=login") == 0) {
        tvs_login_token = 1;
        return;
    }
}

```

```

/* attempt to validate this token with TVS. NOTE: I might
 * well "die()" in the routine as I attempt to handle and
 * token error with a HTTP redirect
 */

/* Note: since tok includes the "TVS=" characters, the
 * token itself starts at tok[4].
 */

tvs_profile = clickshare_attempt_validation(&tok[4], outf);
if (!tvs_profile) {
    /* This token is no good. Forget about it. */
    return;
}

/* Token is OK. Arrange to add it to outgoing URLs. */
clickshare_insert_token_filter (tok, outf);
}

/* -----
 * user has requested CS-protected page without giving a token:
 * redirect them to a CGI script that gives their options
 * -----
 */

void clickshare_auth_required_page (FILE *outf)
{
    char buf[2000], *p;

    /* start: http://myhost/click_cgi/auth_required */
    construct_url (buf, AUTH_REQUIRED_PAGE);

    /* add ?URL= */
    strcat (buf, "?URL=");

    /* add URL, and escape it */
    p = buf + strlen(buf);
    construct_url (p, clickshare_saved_url);
    if (clickshare_saved_query && *clickshare_saved_query) {
        strcat (p, "?");
        strcat (p, clickshare_saved_query);
    }
    escape_url (p);

    die(REDIRECT, buf, outf);
}

/* -----

```

```

* how we "tag" outgoing pages with TVS auth tokens if required
* -----
*/

void clickshare_insert_token_filter (char *tok, FILE *outf)
{
    int pid, fd[2], out;

    /* at this point, we've got the TVS=<token> string in tvs_token,
       it's been removed from the args string */

    out = fileno (outf);

    /* get a pipe: fd[0] is for reading, fd[1] for writing */
    if (pipe (fd) < 0)
        return;

    /* fork */

    pid = fork ();
    if (pid < 0) {
        /* well, not much we can do */
        close (fd[0]);
        close (fd[1]);

        clickshare_dont_shutdown = 0;
        free (tok);
        return;
    }

    if (pid == 0) {
        /* parent: make 'out' refer to fd[1], close fd[0] */
        if (fd[1] != out) {
            dup2 (fd[1], out);
            close (fd[1]);
        }

        close (fd[0]);

        clickshare_dont_shutdown = 1;

        free (tok);
        return;
    }

    /* child: connect fd[0] to stdin, 'out' fd to stdout, close others */
    /* move 'out' out of the way if necessary */
    if (out == 0)
        out = dup (out);

```

```

if (fd[0] != 0) {
    dup2 (fd[0], 0);
    close (fd[0]);
}
if (out != 1) {
    dup2 (out, 1);
    close (out);
}
if (fd[1] > 1)
    close (fd[1]);

{
    int i;
    for (i = 2; i < getdtablesize(); i++)
        close (i);
}

fcntl (0, F_SETFD, 0L);
fcntl (1, F_SETFD, 0L);

/* exec the add-token script */
if (execl (clickshare_add_token_script,
          clickshare_add_token_script, tok, NULL) < 0) {
    /* whoops, we're in trouble */
    exit(-1);
}
}

/* -----
 * handle logging to the Clickshare Transaction Logging Facility
 * -----
 */

int clickshare_open_log () {
    struct sockaddr_un sa;
    int salen;
    char *clickshare_log_path;

    clickshare_log_path = tvs_get_config_param ("LoggerFile");
    if (!clickshare_log_path)
        return 0;

    cs_log_socket = socket(AF_UNIX, SOCK_STREAM, 0);

    if (cs_log_socket < 0)
        return 0;

    bzero((char *) &sa, sizeof(sa));

    sa.sun_family = AF_UNIX;

```

```

strcpy(sa.sun_path, clickshare_log_path);
salen = strlen(sa.sun_path) + sizeof(sa.sun_family);

if (connect(cs_log_socket, (struct sockaddr *)&sa, salen) < 0) {
    close(cs_log_socket);
    cs_log_socket = -1;
    return 0;
}

return 1;
}

/* -----
 * send the "HELLO" to the log facility
 * -----
 */

void clickshare_log_hello() {
    char buf[HUGE_STRING_LEN];
    struct iovec iobuf[2];
    int len, len2;

    if(!clickshare_open_log()) {
        fprintf(stderr, "clickshare_open_log failed\n");
        return;
    }

    if(cs_log_socket < 0)
        return;

    sprintf(buf, "HELLO 0x%08x \"%s\"", tvs_pm_id, tvs_pm_name);

    len = strlen(buf);
    len2 = htonl(len);

    iobuf[0].iov_base = (char *) &len2;
    iobuf[0].iov_len = sizeof(int);
    iobuf[1].iov_base = buf;
    iobuf[1].iov_len = len;

    writev(cs_log_socket, &iobuf[0], 2);
    clickshare_close_log();
}

/* -----
 * log a request out to the central logging facility
 * -----
 */

```

```

void clickshare_log(char *request, char *logmsg) {
    char buf[HUGE_STRING_LEN], *p, *q, *r;
    struct iovec iobuf[2];
    int len, len2;

    if (!tvs_profile)
        return;

    if (!clickshare_open_log()) {
        fprintf(stderr, "clickshare_open_log failed\n");
        return;
    }

    if (cs_log_socket < 0)
        return;

    sprintf (buf,
        "%s user_id=0x%08x pm_id=0x%08x page_class=0x%08x session_id=0x%08x"
        contentpm_id=0x%08x",
        logmsg,
        tvs_get_userid(tvs_profile),
        tvs_get_pmid(tvs_profile),
        clickshare_pageclass,
        tvs_get_sessionid(tvs_profile),
        tvs_pm_id);

    /* strip TVS=token string */
    if (p = strchr(buf, '\\')) {
        /* found URL */
        if (p = strchr(p, '?')) {
            /* found args */
            p++;
            if (!strcmp(p, "TVS=", 4)) {
                /* TVS= is first query argument -- close up */
                q = p + strcspn (p, "\\\"+");
                strcpy(p, q);
            } else if (q = strstr(p, "+TVS=")) {
                /* found TVS=, q points at preceeding '+' delimiter;
                close up */
                r = q + 1 + strcspn (q + 1, "\\\"+");
                strcpy(q, r);
            }
        }
        /* Have we produced something of the form ?+\" or ?\" ? */
        if (*p == '+')
            strcpy(p, p+1);
        if (*p == '\\')
            strcpy(p-1, p);
    }
}

len = strlen(buf);

```



```

len2 = htonl(len);

iobuf[0].iov_base = (char *) &len2;
iobuf[0].iov_len = sizeof(int);
iobuf[1].iov_base = buf;
iobuf[1].iov_len = len;

writev(cs_log_socket, &iobuf[0], 2);
clickshare_close_log();
}

/* -----
 * close up shop (the log anyway)
 * -----
 */

void clickshare_close_log(void) {
    if (cs_log_socket < 0)
        return;

    close(cs_log_socket);
    cs_log_socket = -1;
}

/* -----
 * add Clickshare-specific variables to the environment given to CGI scripts
 * -----
 */

#define MAX_CS_VARS 32 + 16

char **clickshare_add_vars(char **env, FILE *out)
{
    int x;
    char t[HUGE_STRING_LEN];

    if (! (tvsp_profile || clickshare_registrationDB || clickshare_useridDB))
        return env;

    if (! (env = new_env(env, MAX_CS_VARS, &x)))
        die(NO_MEMORY, "add_cgi_vars", out);

    if (tvsp_profile) {
        sprintf(t, "%d", tvsp_pm_id);
        env[x++] = make_env_str("CS_MYOWN_PPID", t, out);

        sprintf(t, "%d", tvsp_get_userid(tvsp_profile));
        env[x++] = make_env_str("CS_USERID", t, out);

        sprintf(t, "%d", tvsp_get_ppid(tvsp_profile));

```

```

env[x++] = make_env_str("CS_P MID", t, out);

sprintf(t, "%08lx", tvs_get_hostid(tvs_profile));
env[x++] = make_env_str("CS_HOSTID", t, out);

sprintf(t, "%d", tvs_get_sessionid(tvs_profile));
env[x++] = make_env_str("CS_SESSIONID", t, out);

sprintf(t, "%d", tvs_get_service_class(tvs_profile));
env[x++] = make_env_str("CS_SERVICE_CLASS", t, out);

sprintf(t, "%d", tvs_get_page_class_limit(tvs_profile));
env[x++] = make_env_str("CS_PAGE_CLASS_LIMIT", t, out);

sprintf(t, "%d", tvs_get_page_count_limit(tvs_profile));
env[x++] = make_env_str("CS_PAGE_COUNT_LIMIT", t, out);

sprintf(t, "%d", tvs_get_service_priority(tvs_profile));
env[x++] = make_env_str("CS_SERVICE_PRIORITY", t, out);

sprintf(t, "%d", tvs_get_customer_group(tvs_profile));
env[x++] = make_env_str("CS_CUSTOMER_GROUP", t, out);

sprintf(t, "%d", tvs_get_adv_context(tvs_profile));
env[x++] = make_env_str("CS_ADV_CONTEXT", t, out);

sprintf(t, "%d", tvs_get_pdac_flag(tvs_profile));
env[x++] = make_env_str("CS_PDAC_FLAG", t, out);

sprintf(t, "%d", tvs_get_privacy1_flag(tvs_profile));
env[x++] = make_env_str("CS_PRIVACY1_FLAG", t, out);

sprintf(t, "%d", tvs_get_premium_flag(tvs_profile));
env[x++] = make_env_str("CS_PREMIUM_FLAG", t, out);
}

if (clickshare_registrationDB)
    env[x++] = make_env_str("CS_REGISTRATION_DB",
        clickshare_registrationDB, out);

if (clickshare_useridDB)
    env[x++] = make_env_str("CS_USERID_DB",
        clickshare_useridDB, out);

env[x] = NULL;
return env;
}

/* -----
* contact the TVS server to invalidate an authentication token.
* -----

```

```

*/

int
clickshare_invalidate_token(char *url, char *args, int in, FILE *outf)
{
    char *tok;

    tok = clickshare_extract_token (args, "TVS=");
    if (!tok) return 0;

#define NO_REASON 0
    if (!tvts_invalidate_token((TVS_TOKEN) tok, NO_REASON))
        return 0;
    else
        return 1;
}

/* -----
 * process a user "logout" - which invalidates his/her authentication token
 * -----
 */

void
clickshare_logout_user(char *url, char *args, int in, FILE *outf)
{
    /* make sure this is a valid user requesting the logout.
     * (clickshare_process_args() will have checked this immediately prior)
     */

    if (!tvts_profile) {
        send_node(LOGOUT_ERROR_PAGE, (char *) NULL, in, outf);
    }

    /* do it */

    if (!clickshare_invalidate_token (url, args, in, outf)) {
        die(SERVER_ERROR, "logout method failed - drop validation error", outf);
    }

    /* send an ack */

    send_node(LOGOUT_THANKYOU_PAGE, (char *) NULL, in, outf);
}

/* -----
 * process a "testdrive" user - a random user ID with minimal privileges
 * -----
 */

void
clickshare_testdrive_user(char *url, char *args, int in, FILE *outf)

```

```

{
    struct timeval tv;
    unsigned int tmp;
    char *tok, p[256];

    /* PS: gonna ignore URL and ARGS for now, but later we may want
     * to feed options in thru here.
     */

    /*
     * first, create a profile for this (random) user
     */

    tvs_profile = tvs_make_testdrive_profile();
    if (!tvs_profile)
        die(NO_MEMORY, "making testdrive profile", outf);

    /* fill in stuff that is non-default */

    tvs_set_pmid(tvs_profile, tvs_pm_id);
    tvs_set_hostid(tvs_profile, inet_addr(remote_ip));
    tvs_set_sessionid(tvs_profile, tvs_make_sessionid(tvs_pm_id, 0));

    /* create a random user ID for the testdrive class (this will give us
     * at least .5M unique user IDs per day, and 4096 per second)
     */

#define RANDOM_USER_MASK    0x10000000
#define TESTDRIVE_USER_CLASS 0x00

#define BIT_MSK2    0x0fff
#define UNBIT_MSK2 12
    tmp = (tv.tv_sec << UNBIT_MSK2) | ((tv.tv_usec >> 4) & BIT_MSK2);
    tmp |= RANDOM_USER_MASK;

    tvs_set_userid(tvs_profile, tmp);

    /*
     * acquire a new authentication token for this user
     */

    tok = tvs_new_token(tvs_profile);
    if (!tok)
        die(SERVER_ERROR, "cannot obtain TVS token for test-drive user", outf);

    /*
     * create the URL of the page to "jump" to
     */

    construct_url(p, TESTDRIVE_JUMP_PAGE);

```

```

escape_url (p);
strcat(p, "?TVS=");
strcat(p, tok);

/* via redirect, user starts his clickshare session as a random user id
 * at the jump page
 */

die (REDIRECT, p, outf);
}

```

Very truly yours,

Steven M. Hoffberg
 Milde & Hoffberg, LLP
 Suite 460
 10 Bank Street
 White Plains, NY 10606
 (914) 949-3100 tel.
 (914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org

Confidentiality Notice: This message, and any attachments thereto, may contain confidential information which is legally privileged. The information is intended only for the use of the intended recipient, generally the individual or entity named above. If you believe you are not the intended recipient, or in the event that this document is received in error, or misdirected, you are requested to immediately inform the sender by reply e-mail at Steve@Hoffberg.org and destroy all copies of the e-mail file and attachments. You are hereby notified that any disclosure, copying, distribution or use of any information contained in this transmission other than by the intended recipient is strictly prohibited.